UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/037,666	01/03/2002	Sriram Vajapeyam	42390P11927	8277

8791 7590 12/23/2005

BLAKELY SOKOLOFF TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1030

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 12/23/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/037,666	VAJAPEYAM ET AL.	
	Examiner	Art Unit	
	David J. Huisman	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *07 October 2005*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-6,8-11,13-15 and 17-33* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-6,8-11,13-15 and 17-33* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *28 December 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-6, 8-11, 13-15, and 17-33 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Amendment as received on 10/7/2005.

Claim Objections

3. Claim 25 recites the limitation "the architectural state". There is insufficient antecedent basis for this limitation in the claim.

Withdrawn Rejections

4. Applicant, by way of amendment, has overcome the prior art rejections set forth in the previous Office Action. Consequently, these rejections are hereby withdrawn by the examiner. However, upon further consideration, a new ground(s) of rejection is applied below.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

6. Claims 1-6, 10-11, 13-15, 20-22, and 27-31 are rejected under 35 U.S.C. 102(b) as being anticipated by Rotenberg et al., "Trace Cache: A Low Latency Approach to High Bandwidth Instruction Fetching," 1996 (herein referred to as Rotenberg).

7. Referring to claim 1, Rotenberg has taught a logic circuit comprising:

a) control flow logic to select and fetch a trace descriptor for processing, the fetched trace descriptor including at least one dependency descriptor, a dependency descriptor including dependency information for an instruction sequence and an address of the instruction sequence. See Fig.4, and section 2.2. Note that the trace descriptor may be at least a trace cache entry (Fig.4, component 25) which includes the components set forth in columns 9-10, including a dependency descriptor (branch flags, branch mask, target address, fall-through address) and an address of the instruction sequence (tag).

b) data flow logic coupled to the control flow logic to execute the instruction sequence according to the dependency information in the dependency descriptor. Note from Fig.4 that instructions, when selected from the trace cache, are sent to a decoder for eventual execution. It should be noted from their descriptions, that the sequence's execution is dependent on the branch flags, branch mask, and target and fall-through addresses, which make up at least a portion of the dependency descriptor.

8. Referring to claim 2, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has further taught a storage area coupled to the control flow logic and the data flow logic, the storage area to store the dependency descriptor from the fetched trace descriptor by the control flow logic. See Fig.4 and note that instructions from the trace cache (which may be considered part of the dependency descriptor) are stored in an instruction latch after being selected by a 2:1 multiplexer. The target/fall through

addresses would also have to be stored so that they may be applied to the caches in the future.

9. Referring to claim 3, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has further taught a storage area coupled to the control flow logic, the storage area to store trace descriptors. See Fig.4, and note the trace cache.

10. Referring to claim 4, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has further taught a storage area coupled to the data flow logic, the storage area to store instructions contiguously based on dependency information. See Fig.4, and note the trace cache.

11. Referring to claim 5, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has further taught a storage area coupled to the data flow logic and control flow logic, the storage area to store live-out data. See the last paragraph in column 1 and note the reference to physical registers being present. Instructions inherently write to registers, and the data stored in the registers may be used, as live-out data, by a subsequent group of instructions.

12. Referring to claim 6, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has further taught a storage area coupled to the control flow logic, the storage area to map live-in and live-out data. See section 4.1 and note the reference to register renaming. Register renaming is the mapping of live-in and live-out data. And, in order to track mappings, there must be storage.

13. Referring to claim 10, Rotenberg has taught a computer system comprising:

a) at least one memory device to store trace descriptors and instruction sequences. See the trace cache of Fig.4.

b) a bus coupled to the at least one memory device. See Fig.4, and note that data must be passed via bus.

c) control flow logic to select and fetch one of the trace descriptors, the fetched trace descriptor including a plurality of dependency descriptors having locations of corresponding instruction sequences and having dependency information for corresponding instruction sequences. See Fig.4, and section 2.2. Note that the trace descriptor may be at least a trace cache entry (Fig.4, component 25) which includes the components set forth in columns 9-10, including dependency descriptors (branch flags, branch mask, and fall-through and target addresses). Note that the descriptors include locations of corresponding instruction sequences (fall-though and target addresses) and dependency information for corresponding instruction sequences (see the branch flags and branch mask).

d) data flow logic coupled to the control flow logic to receive a dependency descriptor dispatched from the control flow logic, to fetch an instruction sequence corresponding to the received dependency descriptor, and to execute the fetched instruction sequence. Again, note the fall through and target addresses. Based on the last branch instruction, a new sequence is fetched from one of these two addresses.

14. Referring to claim 11, Rotenberg has taught a computer system as described in claim 10. Rotenberg has further taught an issue window coupled between the control flow logic and the data flow logic, the issue window to store the dependency descriptor dispatched from the control flow logic. See Fig.4 and note that instructions from the trace cache (which may be considered part of the dependency descriptor) are stored in an

instruction latch after being selected by a 2:1 multiplexer. The latch is storage which issues a window of n instructions.

15. Referring to claim 13, Rotenberg has taught a computer system as described in claim 10. Rotenberg has further taught that at least one memory unit is to store an instruction sequence contiguously based on dependency information. See Fig.4, and note the trace cache.

16. Referring to claim 14, Rotenberg has taught a computer system as described in claim 10. Rotenberg has further taught a storage area coupled to the data flow logic and control flow logic, the storage area to store live-out data. See the last paragraph in column 1 and note the reference to physical registers being present. Instructions inherently write to registers, and the data stored in the registers may be used, as live-out data, by a subsequent group of instructions.

17. Referring to claim 15, Rotenberg has taught a computer system as described in claim 10. Rotenberg has further taught a storage area coupled to the control flow logic, the storage area to map live-in and live-out data. See section 4.1 and note the reference to register renaming. Register renaming is the mapping of live-in and live-out data. And, in order to track mappings, there must be storage.

18. Referring to claim 20, Rotenberg has taught a method of processing instructions comprising:

a) selecting and fetching a trace descriptor in accordance with program control flow. See Fig.4, and section 2.2. Note that the trace descriptor may be at least a trace cache entry (Fig.4, component 25).

b) identifying from the fetched trace descriptor a dependency descriptor including dependency information for a set of instructions and an address of the set of instructions. The trace descriptor would include at least the components set forth in columns 9-10, including a dependency descriptor (branch flags, branch mask, target address, fall-through address) and an address of the instruction sequence (tag).

c) fetching the set of instructions using the address in the dependency descriptor. See Fig.4, and note that along with the control information (branch flags, etc.), the instruction trace/sequence corresponding to the tag (start address) are also fetched.

d) executing the set of instructions according to the dependency information in the dependency descriptor. Note that the sequence is executed based on the branch flags/mask.

19. Referring to claim 21, Rotenberg has taught a method as described in claim 20. Rotenberg has further taught updating live-out data in a storage area. See the last paragraph in column 1 and note the reference to physical registers being present. Instructions inherently write to registers, and the data stored in the registers may be used, as live-out data, by a subsequent group of instructions.

20. Referring to claim 22, Rotenberg has taught a method as described in claim 20. Rotenberg has further taught:

a) storing the identified dependency descriptor from a control flow logic into a storage area. See Fig.4 and note that instructions from the trace cache (which may be considered part of the dependency descriptor) are stored in an instruction latch after being selected by a 2:1 multiplexer. The target/fall through addresses would also have to be stored so that they may be applied to the caches in the future.

b) reading the dependency descriptor out of the storage area into the data flow logic. Ultimately, the instruction needs to be executed so it will be read from the latch. Similarly, the target/fall-through address will need to be read and applied to the cache.

21. Referring to claim 27, Rotenberg has taught a method as described in claim 20. Rotenberg has further taught that the selecting comprises predicting a next trace descriptor to process. See columns 9-10, and note the target and fall-through addresses. The last branch in the current trace is predicted. If it's predicted taken, then the next trace descriptor to be selected and processed would be the trace descriptor at the target address. Otherwise, it would be the descriptor at the fall-through address.

22. Referring to claim 28, Rotenberg has taught a machine-readable medium that provides instructions, which when executed by a machine cause the machine to perform operations comprising:

a) selecting and fetching a trace descriptor in accordance with program control flow. See Fig.4, and section 2.2. Note that the trace descriptor may be at least a trace cache entry (Fig.4, component 25).

b) identifying from the fetched trace descriptor a dependency descriptor including dependency information for a set of instructions and an address of the set of instructions. The trace descriptor would include at least the components set forth in columns 9-10, including a dependency descriptor (branch flags, branch mask, target address, fall-through address) and an address of the instruction sequence (tag).

c) fetching the set of instructions using the address in the dependency descriptor. See Fig.4, and note that along with the control information (branch flags, etc.), the instruction trace/sequence corresponding to the tag (start address) are also fetched.

d) executing the set of instructions according to the dependency information in the dependency descriptor. Note that the sequence is executed based on the branch flags/mask.

23. Referring to claim 29, Rotenberg has taught a medium as described in claim 28. Rotenberg has further taught that the operations further comprise updating live-out data in a storage area. See the last paragraph in column 1 and note the reference to physical registers being present. Instructions inherently write to registers, and the data stored in the registers may be used, as live-out data, by a subsequent group of instructions.

24. Referring to claim 30, Rotenberg has taught a medium as described in claim 28. Rotenberg has further taught:

a) storing the dependency descriptor in an issue window by control flow logic. See Fig.4 and note that instructions from the trace cache (which may be considered part of the dependency descriptor) are stored in an instruction latch after being selected by a 2:1 multiplexer. The target/fall through addresses would also have to be stored so that they may be applied to the caches in the future.

b) reading the dependency descriptor out of the issue window into data flow logic. Ultimately, the instruction needs to be executed so it will be read from the latch. Similarly, the target/fall-through address will need to be read and applied to the cache.

25. Referring to claim 31, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has further taught that the fetched trace descriptor includes a plurality of dependency descriptors having addresses of corresponding instruction sequences and having dependency information for corresponding instruction sequences. See Fig.4, and section 2.2. Note that the trace descriptor may be at least a trace cache entry (Fig.4,

component 25) which includes the components set forth in columns 9-10, including dependency descriptors (branch flags, branch mask, and fall-through and target addresses). Note that the descriptors include locations of corresponding instruction sequences (fall-though and target addresses) and dependency information for corresponding instruction sequences (see the branch flags and branch mask).

Claim Rejections - 35 USC § 103

26. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

27. Claims 8-9, 17-19, and 32-33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, as applied above, in view of Vajapeyam et al., "Improving Superscalar Instruction Dispatch and Issue by Exploiting Dynamic Code Sequences," 1997 (as disclosed by applicant and herein referred to as Vajapeyam).

28. Referring to claim 8, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has not taught that the trace descriptor includes aggregate live-in data for a plurality of dependency descriptors in the trace descriptor. However, Vajapeyam has taught that trace cache entries can include live-in data in order to exploit the fact that not all registers are live beyond the trace that produces them. Such exploitation can reduce bottlenecks. See section 2.2. As a result, it would have been obvious to modify Rotenberg's trace cache such that the entries also contain live-in data bits to help reduce bottlenecks.

29. Referring to claim 9, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has not taught that the trace descriptor includes aggregate live-out data for a plurality of dependency descriptors in the trace descriptor. However, Vajapeyam has taught that trace cache entries can include live-out data in order to exploit the fact that not all registers are live beyond the trace that produces them. Such exploitation can reduce bottlenecks. See section 2.2. As a result, it would have been obvious to modify Rotenberg's trace cache such that the entries also contain live-out data bits to help reduce bottlenecks.

30. Referring to claim 17, Rotenberg has taught a computer system as described in claim 10. Rotenberg has not taught that the fetched trace descriptor includes aggregate live-in data for dependency descriptors in the fetched trace descriptor. However, Vajapeyam has taught that trace cache entries can include live-in data in order to exploit the fact that not all registers are live beyond the trace that produces them. Such exploitation can reduce bottlenecks. See section 2.2. As a result, it would have been obvious to modify Rotenberg's trace cache such that the entries also contain live-in data bits to help reduce bottlenecks.

31. Referring to claim 18, Rotenberg has taught a computer system as described in claim 10. Rotenberg has not taught that the fetched trace descriptor includes aggregate live-out data for dependency descriptors in the fetched trace descriptor. However, Vajapeyam has taught that trace cache entries can include live-out data in order to exploit the fact that not all registers are live beyond the trace that produces them. Such exploitation can reduce bottlenecks. See section 2.2. As a result, it would have been

obvious to modify Rotenberg's trace cache such that the entries also contain live-out data bits to help reduce bottlenecks.

32. Referring to claim 19, Rotenberg has taught a computer system as described in claim 10. Rotenberg has not taught that dependency information of the received dependency descriptor includes live-out data. However, Vajapeyam has taught that trace cache entries can include live in and live-out data in order to exploit the fact that not all registers are live beyond the trace that produces them. Such exploitation can reduce bottlenecks. See section 2.2. As a result, it would have been obvious to modify Rotenberg's trace cache such that the entries also contain live-in and live-out data bits to help reduce bottlenecks.

33. Referring to claim 32, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has not taught that the dependency information includes live-in data. However, Vajapeyam has taught that trace cache entries can include live-in data in order to exploit the fact that not all registers are live beyond the trace that produces them. Such exploitation can reduce bottlenecks. See section 2.2. As a result, it would have been obvious to modify Rotenberg's trace cache such that the entries also contain live-in data bits to help reduce bottlenecks.

34. Referring to claim 33, Rotenberg has taught a logic circuit as described in claim 1. Rotenberg has not taught that the dependency information includes live-out data. However, Vajapeyam has taught that trace cache entries can include live-out data in order to exploit the fact that not all registers are live beyond the trace that produces them. Such exploitation can reduce bottlenecks. See section 2.2. As a result, it would have been

obvious to modify Rotenberg's trace cache such that the entries also contain live-out data bits to help reduce bottlenecks.

35. Claims 23-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, as applied above, in view of Arimilli et al., U.S. Patent No. 6,427,204 (as applied in the previous Office Action and herein referred to as Arimilli).

36. Referring to claim 23, Rotenberg has taught a method as described in claim 20. Rotenberg has not taught that the fetching of a set of instructions is completed just in time for execution. However, Arimilli has taught such a concept. See column 3, lines 1-17. Note that Arimilli has taught that this is a more efficient way of fetching because instructions are only delivered when they are actually needed and pipeline bubbles are prevented. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Rotenberg such that instructions are fetched just-in-time, as taught by Arimilli.

37. Referring to claim 24, Rotenberg has taught a method as described in claim 20. Although Rotenberg has not taught that the instructions are out of order, Arimilli has taught such a concept. See column 1, line 61, to column 2, line 6. Note that the use of resources and efficiency are maximized with out-of-order execution. In addition, out-of-order execution allows for a reduction in stalling. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Rotenberg to include instructions that are out-of-order, as taught by Arimilli.

38. Claims 25-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rotenberg, as applied above, in view of Witt et al., U.S. Patent No. 6,018,798 (as applied in the previous Office Action and herein referred to as Witt).

39. Referring to claim 25, Rotenberg has taught a method as described in claim 21. Rotenberg has not explicitly taught updating the architectural state using the data in the storage area. However, Witt has taught the concept of having a speculative register file (future file 88, Fig.3) and an actual register file (Fig.3, component 102). The speculative register file holds the most current state of the machine (values determined via speculative execution) and by doing this, instructions may be executed speculatively. Once it is determined that instructions are no longer speculative, the speculative results are made architectural results by writing them to the actual register file. See column 12, line 66, to column 13, line 45. This is a known concept in the art. In essence, this scheme allows for speculative execution which is a method of executing instructions before it is known that they should execute (they are predicted to execute). This maximizes efficiency if they indeed were to execute (predicted correctly). As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Rotenberg such that the architectural state is updated using the data in the speculative storage.

40. Referring to claim 26, Rotenberg in view of Witt has taught a method as described in claim 25. Witt has further taught recovering an earlier architectural state after a misprediction using data in the storage area. See column 18, lines 54-67, and note that after a misprediction, a previous state is achieved by copying actual values into the

future file (so that the speculative values are correct). Consequently, by using this newly written data, the system recovers an earlier architectural state.

Conclusion

41. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Applicant is reminded that in amending in response to a rejection of claims, the patentable novelty must be clearly shown in view of the state of the art disclosed by the references cited and the objections made. Applicant must also show how the amendments avoid such references and objections. See 37 CFR § 1.111(c).

Bala, U.S. Patent No. 6,351,844, has taught a method for selecting active code traces for translation in a caching dynamic translator. The method includes reading

branch history (dependency information) from a trace entry including the history and a sequence start address.

Sinharoy, U.S. Patent No. 6,247,097, has taught an aligned instruction cache handling of instruction fetches across multiple predicted branch instructions. The caching system includes entries having start addresses and a variety of dependency information.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

DJH
David J. Huisman
December 19, 2005